

Índice

1. algorithm	2	6.9. Extended Euclid	15
2. Estructuras	2	6.10. Polinomio	15
2.1. Easy segment	2	6.11. FFT	16
2.2. RMQ (static)	2	7. Grafos	16
2.3. RMQ (dynamic)	2	7.1. Bellman-Ford	16
2.4. RMQ (lazy)	3	7.2. 2-SAT + Tarjan SCC	16
2.5. RMQ (persistente)	3	7.3. Puentes y Articulation Points	17
2.6. Union Find	4	7.4. LCA + Climb	17
2.7. Disjoint Intervals	4	7.5. Heavy Light Decomposition	18
2.8. RMQ (2D)	4	7.6. Centroid Decomposition	18
2.9. Treap para set	4	7.7. Euler Cycle	18
2.10. Treap para arreglo	5	7.8. Chu-liu	18
2.11. Set con busq binaria	5	7.9. Hungarian	19
3. Algos	5	7.10. Dynamic Conectivity	20
3.1. Longest Increasing Subsequence	5	8. Network Flow	20
3.2. Alpha-Beta pruning	6	8.1. Dinic	20
3.3. Mo's algorithm	6	8.2. Edmonds Karp's	21
3.4. Ternary search	6	8.3. Max Matching	21
4. Strings	6	8.4. Min-cost Max-flow	21
4.1. Manacher	6	9. Template y Otros	22
4.2. KMP	6		
4.3. Trie	7		
4.4. Suffix Array (largo, nlogn)	7		
4.5. String Matching With Suffix Array	7		
4.6. LCP (Longest Common Prefix)	7		
4.7. Corasick	7		
4.8. Suffix Automaton	8		
4.9. Z Function	8		
4.10. Palindromic tree	9		
4.11. Rabin Karp - Distinct Substrings	9		
5. Geometria	9		
5.1. Punto	9		
5.2. Orden radial de puntos	10		
5.3. Line	10		
5.4. Segment	10		
5.5. Polygon Area	10		
5.6. Circle	10		
5.7. Point in Poly	11		
5.8. Point in Convex Poly log(n)	11		
5.9. Convex Check CHECK	11		
5.10. Convex Hull	12		
5.11. Cut Polygon	12		
5.12. Bresenham	12		
5.13. Interseccion de Circulos en $n^3 \log(n)$	12		
6. Math	13		
6.1. Identidades	13		
6.2. Ec. Caracteristica	13		
6.3. Combinatorio	13		
6.4. Gauss Jordan, Determinante	13		
6.5. Teorema Chino del Resto	14		
6.6. Funciones de primos	14		
6.7. Phollard's Rho (rolando)	14		
6.8. GCD	15		

1. algorithm

```
#include <algorithm> #include <numeric>
```

Algo	Funcion
sort, stable_sort	ordena el intervalo
nth_element	void ordena el n-esimo, y particiona el resto
fill, fill_n	void llena [f, l) o [f, f+n) con elem
lower_bound, upper_bound	it al primer / ultimo donde se puede insertar elem para que quede ordenada
binary_search	bool esta elem en [f, l)
copy	hace resul+i=f+i $\forall i$
find, find_if, find_first_of	it encuentra i $\in [f, l)$ tq. i=elem, pred(i), i $\in [f2, l2)$
count, count_if	cuenta elem, pred(i)
search	busca [f2, l2) $\in [f, l)$
replace, replace_if	cambia old / pred(i) por new / pred, new
reverse	da vuelta
partition, stable_partition	pred(i) ad, !pred(i) atras
min_element, max_element	it min, max de [f, l)
lexicographical_compare	bool con [f1, l1]; [f2, l2]
next/prev_permutation	deja en [f, l) la perm sig, ant
set_intersection, set_difference, set_union, set_symmetric_difference,	[res, ...) la op. de conj
push_heap, pop_heap, make_heap	mete/saca e en heap [f, l), hace un heap de [f, l)
is_heap	bool es [f, l) un heap
accumulate	$T = \sum$ /oper de [f, l)
inner_product	$T = i + [f1, l1) \cdot [f2, \dots)$
partial_sum	$r+i = \sum$ /oper de [f, f+i] $\forall i \in [f, l)$
__builtin_ffs	Pos. del primer 1 desde la derecha
__builtin_clz	Cant. de ceros desde la izquierda.
__builtin_ctz	Cant. de ceros desde la derecha.
__builtin_popcount	Cant. de 1's en x.
__builtin_parity	1 si x es par, 0 si es impar.
__builtin_XXXXXXll	= pero para long's.

2. Estructuras

2.1. Easy segment

```
1 const int N = 1e5; // limit for size
2 int n; // array size
3 int t[2 * N];
4
5 void build() {
6     for (int i = n - 1; i > 0; --i)
7         t[i] = t[i<<1] + t[i<<1|1];
8 }
9
10 void modify(int p, int value) {
11     for (t[p += n] = value; p > 1; p >>= 1)
12         t[p>>1] = t[p] + t[p^1];
13 }
14
15 int query(int l, int r) {
16     int res = 0;
17     for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
18         if (l&1) res += t[l++];
19         if (r&1) res += t[--r];
20     }
21     return res;
22 }
23
24 int main() {
25     scanf("%d", &n);
26     for (int i = 0; i < n; ++i)
27         scanf("%d", t + n + i);
28     build();
29     modify(0, 1);
30     printf("%d\n", query(3, 11));
31     return 0;
32 }
```

2.2. RMQ (static)

Dado un arreglo y una operacion asociativa *idempotente*, get(i, j) opera sobre el rango [i, j). Restriccion: $LVL \geq \text{ceil}(\log n)$; Usar [] para llenar arreglo y luego build().

```
1 struct RMQ{
2     #define LVL 10
3     tipo vec[LVL][1<<(LVL+1)];
4     tipo &operator[] (int p){return vec[0][p];}
5     tipo get(int i, int j) { //intervalo [i,j)
6         int p = 31-__builtin_clz(j-i);
7         return min(vec[p][i], vec[p][j-(1<<p)]);
8     }
9     void build(int n) { //O(nlogn)
10         int mp = 31-__builtin_clz(n);
11         forn(p, mp) forn(x, n-(1<<p))
12             vec[p+1][x] = min(vec[p][x], vec[p][x+(1<<p)]);
13     }
14 }
```

2.3. RMQ (dynamic)

```

1 //Dado un arreglo y una operacion asociativa con
  neutro, get(i, j) opera sobre el rango [i, j)
  .
2 #define MAXN 100000
3 #define operacion(x, y) max(x, y)
4 const int neutro=0;
5 struct RMQ{
6     int sz;
7     tipo t[4*MAXN];
8     tipo &operator[] (int p){return t[sz+p];}
9     void init(int n){//O(nlgn)
10         sz = 1 << (32-__builtin_clz(n));
11         forn(i, 2*sz) t[i]=neutro;
12     }
13     void updall(){//O(n)
14         dforn(i, sz) t[i]=operacion(t[2*i], t[2*i+1])
15             ;}
16     tipo get(int i, int j){return get(i,j,1,0,sz);}
17     tipo get(int i, int j, int n, int a, int b){//O
18         (lgn)
19         if(j<=a || i>=b) return neutro;
20         if(i<=a && b<=j) return t[n];
21         int c=(a+b)/2;
22         return operacion(get(i, j, 2*n, a, c), get(i,
23             j, 2*n+1, c, b));
24     }
25     void set(int p, tipo val){//O(lgn)
26         for(p+=sz; p>0 && t[p]!=val;){
27             t[p]=val;
28             p/=2;
29             val=operacion(t[p*2], t[p*2+1]);
30         }
31     }
32 }rmq;
33 //Usage:
34 cin >> n; rmq.init(n); forn(i, n) cin >> rmq[i];
35 rmq.updall();

```

2.4. RMQ (lazy)

```

1 //Dado un arreglo y una operacion asociativa con
  neutro, get(i, j) opera sobre el rango [i, j)
  .
2 typedef int Elem;//Elem de los elementos del
  arreglo
3 typedef int Alt;//Elem de la alteracion
4 #define operacion(x,y) x+y
5 const Elem neutro=0; const Alt neutro2=0;
6 #define MAXN 100000
7 struct RMQ{
8     int sz;
9     Elem t[4*MAXN];
10     Alt dirty[4*MAXN]; //las alteraciones pueden ser
11     de distinto Elem
12     Elem &operator[] (int p){return t[sz+p];}
13     void init(int n){//O(nlgn)
14         sz = 1 << (32-__builtin_clz(n));
15         forn(i, 2*sz) t[i]=neutro;
16         forn(i, 2*sz) dirty[i]=neutro2;
17     }
18 }

```

```

17 void push(int n, int a, int b){//propaga el
18     dirty a sus hijos
19     if(dirty[n]!=0){
20         t[n]+=dirty[n]*(b-a); //altera el nodo
21         if(n<sz){
22             dirty[2*n]+=dirty[n];
23             dirty[2*n+1]+=dirty[n];
24         }
25         dirty[n]=0;
26     }
27     Elem get(int i, int j, int n, int a, int b){//O
28         (lgn)
29         if(j<=a || i>=b) return neutro;
30         push(n, a, b); //corrige el valor antes de
31         usarlo
32         if(i<=a && b<=j) return t[n];
33         int c=(a+b)/2;
34         return operacion(get(i, j, 2*n, a, c), get(i,
35             j, 2*n+1, c, b));
36     }
37     Elem get(int i, int j){return get(i,j,1,0,sz);}
38     //altera los valores en [i, j) con una
39     alteracion de val
40     void alterar(Alt val, int i, int j, int n, int
41         a, int b){//O(lgn)
42         push(n, a, b);
43         if(j<=a || i>=b) return;
44         if(i<=a && b<=j){
45             dirty[n]+=val;
46             push(n, a, b);
47             return;
48         }
49         int c=(a+b)/2;
50         alterar(val, i, j, 2*n, a, c), alterar(val, i
51             , j, 2*n+1, c, b);
52         t[n]=operacion(t[2*n], t[2*n+1]); //por esto
53         es el push de arriba
54     }
55     void alterar(Alt val, int i, int j){alterar(val
56         ,i,j,1,0,sz);}
57 }rmq;

```

2.5. RMQ (persistente)

```

1 typedef int tipo;
2 tipo oper(const tipo &a, const tipo &b){
3     return a+b;
4 }
5 struct node{
6     tipo v; node *l,*r;
7     node(tipo v):v(v), l(NULL), r(NULL) {}
8     node(node *l, node *r) : l(l), r(r){
9         if(!l) v=r->v;
10        else if(!r) v=l->v;
11        else v=oper(l->v, r->v);
12    }
13 };
14 node *build (tipo *a, int tl, int tr) { //
15     modificar para que tome tipo a

```

```

15     if (tl+1==tr) return new node(a[tl]);
16     int tm=(tl + tr)>>1;
17     return new node(build(a, tl, tm), build(a, tm,
18         tr));
19 }
20 node *update(int pos, int new_val, node *t, int
21     tl, int tr){
22     if (tl+1==tr) return new node(new_val);
23     int tm=(tl+tr)>>1;
24     if(pos < tm) return new node(update(pos,
25         new_val, t->l, tl, tm), t->r);
26     else return new node(t->l, update(pos, new_val,
27         t->r, tm, tr));
28 }
29 tipo get(int l, int r, node *t, int tl, int tr){
30     if(l==tl && tr==r) return t->v;
31     int tm=(tl + tr)>>1;
32     if(r<=tm) return get(l, r, t->l, tl, tm);
33     else if(l>=tm) return get(l, r, t->r, tm, tr)
34         ;
35     return oper(get(l, tm, t->l, tl, tm), get(tm, r
36         , t->r, tm, tr));
37 }

```

2.6. Union Find

```

1 class UnionFind {
2 private:
3     vi p, rank, setSize;
4     int numSets;
5 public:
6     UnionFind(int N) {
7         setSize.assign(N, 1); numSets = N; rank.
8             assign(N, 0);
9         p.assign(N, 0); for (int i = 0; i < N; i++) p
10             [i] = i; }
11     int findSet(int i) { return (p[i] == i) ? i : (
12         p[i] = findSet(p[i])); }
13     bool isSameSet(int i, int j) { return findSet(i
14         ) == findSet(j); }
15     void unionSet(int i, int j) {
16         if (!isSameSet(i, j)) { numSets--;
17             int x = findSet(i), y = findSet(j);
18             // rank is used to keep the tree short
19             if (rank[x] > rank[y]) { p[y] = x; setSize[x]
20                 += setSize[y]; }
21             else { p[x] = y; setSize[y]
22                 += setSize[x];
23                 if (rank[x] == rank[
24                     y]) rank[y]++; }
25         }
26     }
27     int numDisjointSets() { return numSets; }
28     int sizeOfSet(int i) { return setSize[findSet(i
29         )]; }
30 };

```

2.7. Disjoint Intervals

```

1 bool operator< (const ii &a, const ii &b) {return
2     a.fst<b.fst;}
3 //Stores intervals as [first, second]

```

```

3 //in case of a collision it joins them in a
4     single interval
5 struct disjoint_intervals {
6     set<ii> segs;
7     void insert(ii v) {//O(lgn)
8         if(v.snd-v.fst==0.) return;//OJO
9         set<ii>::iterator it,at;
10         at = it = segs.lower_bound(v);
11         if (at!=segs.begin() && (--at)->snd >= v.fst)
12             v.fst = at->fst, --it;
13         for(; it!=segs.end() && it->fst <= v.snd;
14             segs.erase(it++))
15             v.snd=max(v.snd, it->snd);
16         segs.insert(v);
17     }
18 };

```

2.8. RMQ (2D)

```

1 struct RMQ2D{//n filas x m columnas
2     int sz;
3     RMQ t[4*MAXN];
4     RMQ &operator[] (int p){return t[sz/2+p];};//t[i
5         ][j]=i fila, j col
6     void init(int n, int m){//O(n*m)
7         sz = 1 << (32-__builtin_clz(n));
8         forn(i, 2*sz) t[i].init(m); }
9     void set(int i, int j, tipo val){//O(lgm.lgn)
10         for(i+=sz; i>0;){
11             t[i].set(j, val);
12             i/=2;
13             val=operacion(t[i*2][j], t[i*2+1][j]);
14         } }
15     tipo get(int i1, int j1, int i2, int j2){return
16         get(i1,j1,i2,j2,1,0,sz);}
17     //O(lgm.lgn), rangos cerrado abierto
18     int get(int i1, int j1, int i2, int j2, int n,
19         int a, int b){
20         if(i2<=a || i1>=b) return 0;
21         if(i1<=a && b<=i2) return t[n].get(j1, j2);
22         int c=(a+b)/2;
23         return operacion(get(i1, j1, i2, j2, 2*n, a,
24             c),
25             get(i1, j1, i2, j2, 2*n+1, c, b));
26     }
27 } rmq;
28 //Example to initialize a grid of M rows and N
29     columns:
30 RMQ2D rmq; rmq.init(n,m);
31 forn(i, n) forn(j, m){
32     int v; cin >> v; rmq.set(i, j, v);}

```

2.9. Treap para set

Treap para set tiene un Key unico por nodo. En el split if (key <= t->key). En at, if(key == t->key) return t; en lugar de pos.

```

1 void erase(pnode &t, Key key) {
2     if (!t) return; push(t);
3     if (key == t->key) t=merge(t->l, t->r);
4     else if (key < t->key) erase(t->l, key);

```

```

5     else erase(t->r, key);
6     if(t) pull(t);}

```

2.10. Treap para arreglo

```

1  typedef struct node *pnode;
2  struct node{
3      Value val, mini;
4      int dirty;
5      int prior, size;
6      pnode l,r,parent;
7      node(Value val): val(val), mini(val), dirty
          (0), prior(rand()), size(1), l(0), r(0),
          parent(0) {}
8  };
9  static int size(pnode p) { return p ? p->size :
10     0; }
11 void push(pnode p) { //propagar dirty a los hijos(
12     aca para lazy)
13     p->val.fst+=p->dirty;
14     p->mini.fst+=p->dirty;
15     if(p->l) p->l->dirty+=p->dirty;
16     if(p->r) p->r->dirty+=p->dirty;
17     p->dirty=0;
18 }
19 static Value mini(pnode p) { return p ? push(p),
20     p->mini : ii(1e9, -1); }
21 // Update function and size from children's Value
22 void pull(pnode p) { //recalcular valor del nodo
23     aca (para rmq)
24     p->size = 1 + size(p->l) + size(p->r);
25     p->mini = min(min(p->val, mini(p->l)), mini(p->
26     r)); //operacion del rmq!
27     p->parent=0;
28     if(p->l) p->l->parent=p;
29     if(p->r) p->r->parent=p;
30 }
31 //junta dos arreglos
32 pnode merge(pnode l, pnode r) {
33     if (!l || !r) return l ? l : r;
34     push(l), push(r);
35     pnode t;
36     if (l->prior < r->prior) l->r=merge(l->r, r), t
37         = l;
38     else r->l=merge(l, r->l), t = r;
39     pull(t);
40     return t;
41 }
42 //parte el arreglo en dos, sz(l)==tam
43 void split(pnode t, int tam, pnode &l, pnode &r)
44 {
45     if (!t) return void(l = r = 0);
46     push(t);
47     if (tam <= size(t->l)) split(t->l, tam, l, t->l
48         ), r = t;
49     else split(t->r, tam - 1 - size(t->l), t->r, r)
50         , l = t;
51     pull(t);
52 }

```

```

44 pnode at(pnode t, int pos) {
45     if(!t) exit(1);
46     push(t);
47     if(pos == size(t->l)) return t;
48     if(pos < size(t->l)) return at(t->l, pos);
49     return at(t->r, pos - 1 - size(t->l));
50 }
51 int getpos(pnode t){ //inversa de at
52     if(!t->parent) return size(t->l);
53     if(t==t->parent->l) return getpos(t->parent)-
54         size(t->r)-1;
55     return getpos(t->parent)+size(t->l)+1;
56 }
57 void split(pnode t, int i, int j, pnode &l, pnode
58     &m, pnode &r) {
59     split(t, i, l, t), split(t, j-i, m, r);}
60 Value get(pnode &p, int i, int j){ //like rmq
61     pnode l,m,r;
62     split(p, i, j, l, m, r);
63     Value ret=mini(m);
64     p=merge(l, merge(m, r));
65     return ret;
66 }
67 //Sample program: C. LCA Online from Petrozavodsk
68     Summer-2012. Petrozavodsk SU Contest
69 //Available at http://opentrains.snarknews.info/~
70     ejudge
71 const int MAXN=300100;
72 int n;
73 pnode beg[MAXN], fin[MAXN];
74 pnode lista;

```

2.11. Set con busq binaria

```

1  #include <ext/pb_ds/assoc_container.hpp>
2  #include <ext/pb_ds/tree_policy.hpp>
3  using namespace __gnu_pbds;
4  typedef tree<int,null_type,less<int>, //key,mapped
5      type, comparator
6      rb_tree_tag,tree_order_statistics_node_update
7      > set_t;
8  //find_by_order(i) devuelve iterador al i-esimo
9      elemento
10 //order_of_key(k): devuelve la pos del lower
11     bound de k

```

3. Algos

3.1. Longest Increasing Subsequence

```

1
2  typedef vector<int> VI;
3  typedef pair<int,int> PII;
4  typedef vector<PII> VPII;
5
6  #define STRICTLY_INCREASNG
7
8  VI LongestIncreasingSubsequence(VI v) {
9      VPII best;

```

```

10 VI dad(v.size(), -1);
11
12 for (int i = 0; i < v.size(); i++) {
13 #ifdef STRICTLY_INCREASNG
14     PII item = make_pair(v[i], 0);
15     VPII::iterator it = lower_bound(best.begin(),
16         best.end(), item);
17     item.second = i;
18 #else
19     PII item = make_pair(v[i], i);
20     VPII::iterator it = upper_bound(best.begin(),
21         best.end(), item);
22 #endif
23     if (it == best.end()) {
24         dad[i] = (best.size() == 0 ? -1 : best.back
25             ().second);
26         best.push_back(item);
27     } else {
28         dad[i] = it == best.begin() ? -1 : prev(it)
29             ->second;
30         *it = item;
31     }
32 }

```

3.2. Alpha-Beta pruning

```

1 ll alphabeta(State &s, bool player = true, int
2     depth = 1e9, ll alpha = -INF, ll beta = INF)
3     { //player = true -> Maximiza
4     if(s.isFinal()) return s.score;
5     //~ if (!depth) return s.heuristic();
6     vector<State> children;
7     s.expand(player, children);
8     int n = children.size();
9     forn(i, n) {
10         ll v = alphabeta(children[i], !player,
11             depth-1, alpha, beta);
12         if(!player) alpha = max(alpha, v);
13         else beta = min(beta, v);
14         if(beta <= alpha) break;
15     }
16     return !player ? alpha : beta;

```

3.3. Mo's algorithm

```

1 int n,sq;
2 struct Qu{ //queries [l, r]
3     //intervalos cerrado abiertos !!! importante
4     int l, r, id;
5 }qs[MAXN];
6 int ans[MAXN], curans; //ans[i]=ans to ith query
7 bool bymos(const Qu &a, const Qu &b){
8     if(a.l/sq!=b.l/sq) return a.l<b.l;
9     return (a.l/sq)&1? a.r<b.r : a.r>b.r;
10 }
11 void mos(){
12     forn(i, t) qs[i].id=i;
13     sort(qs, qs+t, bymos);
14     int cl=0, cr=0;
15     sq=sqrt(n);

```

```

16     curans=0;
17     forn(i, t){ //intervalos cerrado abiertos !!!
18         importante!!
19         Qu &q=qs[i];
20         while(cl>q.l) add(--cl);
21         while(cr<q.r) add(cr++);
22         while(cl<q.l) remove(cl++);
23         while(cr>q.r) remove(--cr);
24         ans[q.id]=curans;
25     }

```

3.4. Ternary search

```

1 #include <functional>
2 //Retorna argmax de una funcion unimodal 'f' en
3 //el rango [left,right]
4 double ternarySearch(double l, double r, function
5     <double(double)> f){
6     for(int i = 0; i < 300; i++){
7         double m1 = l+(r-l)/3, m2 = r-(r-l)/3;
8         if (f(m1) < f(m2)) l = m1; else r = m2;
9     }
10     return (left + right)/2;

```

4. Strings

4.1. Manacher

```

1 int d1[MAXN]; //d1[i]=long del maximo palindromo
2 //impar con centro en i
3 int d2[MAXN]; //d2[i]=analogo pero para longitud
4 //par
5 //0 1 2 3 4
6 //a a b c c <--d1[2]=3
7 //a a b b <--d2[2]=2 (están uno antes)
8 void manacher(){
9     int l=0, r=-1, n=sz(s);
10     forn(i, n){
11         int k=(i>r? 1 : min(d1[l+r-i], r-i));
12         while(i+k<n && i-k>=0 && s[i+k]==s[i-k]) ++k;
13         d1[i] = k--;
14         if(i+k > r) l=i-k, r=i+k;
15     }
16     l=0, r=-1;
17     forn(i, n){
18         int k=(i>r? 0 : min(d2[l+r-i+1], r-i+1))+1;
19         while(i+k-1<n && i-k>=0 && s[i+k-1]==s[i-k])
20             ++k;
21         d2[i] = --k;
22         if(i+k-1 > r) l=i-k, r=i+k-1;
23     }

```

4.2. KMP

```

1 string T; //cadena donde buscar(what)
2 string P; //cadena a buscar(what)
3 int b[MAXLEN]; //back table b[i] maximo borde de
4 // [0..i)
5 void kmppre(){ //by gabina with love

```

```

5     int i=0, j=-1; b[0]=-1;
6     while(i<sz(P)){
7         while(j>=0 && P[i] != P[j]) j=b[j];
8         i++, j++, b[i] = j;
9     }
10 }
11 void kmp(){
12     int i=0, j=0;
13     while(i<sz(T)){
14         while(j>=0 && T[i]!=P[j]) j=b[j];
15         i++, j++;
16         if(j==sz(P)) printf("P_is_found_at_index_
17                             %d_in_T\n", i-j), j=b[j];
18     }
19 }
20 int main(){
21     cout << "T=";
22     cin >> T;
23     cout << "P=";

```

4.3. Trie

```

1 struct trie{ map<char, trie> m;
2     void add(const string &s, int p=0){ if(s[p]) m[
3         s[p]].add(s, p+1);}
4     void dfs(){/*Do stuff*/ forall(it, m) it->
5         second.dfs();}};

```

4.4. Suffix Array (largo, nlogn)

```

1 #define MAX_N 1000
2 #define rBOUND(x) (x<n? r[x] : 0)
3 //sa will hold the suffixes in order.
4 int sa[MAX_N], r[MAX_N], n;
5 string s; //input string, n=sz(s)
6
7 int f[MAX_N], tmpsa[MAX_N];
8 void countingSort(int k){
9     zero(f);
10    forn(i, n) f[rBOUND(i+k)]++;
11    int sum=0;
12    forn(i, max(255, n)){
13        int t=f[i]; f[i]=sum; sum+=t;}
14    forn(i, n)
15        tmpsa[f[rBOUND(sa[i]+k)]++] = sa[i];
16    memcpy(sa, tmpsa, sizeof(sa));
17 }
18 void constructsa(){//O(n log n)
19     n=sz(s);
20     forn(i, n) sa[i]=i, r[i]=s[i];
21     for(int k=1; k<n; k<=1){
22         countingSort(k), countingSort(0);
23         int rank, tmpr[MAX_N];
24         tmpr[sa[0]]=rank=0;
25         forr(i, 1, n)
26             tmpr[sa[i]]=(r[sa[i]]==r[sa[i-1]] && r[sa[i]
27                 +k]==r[sa[i-1]+k])? rank : ++rank;
28         memcpy(r, tmpr, sizeof(r));
29         if(r[sa[n-1]]==n-1) break;
30     }

```

```

30 }
31 //returns (lowerbound, upperbound) of the search
32 ii stringMatching(string P){ //O(sz(P)lgn)
33     int lo=0, hi=n-1, mid=lo;
34     while(lo<hi){

```

4.5. String Matching With Suffix Array

```

1 //returns (lowerbound, upperbound) of the search
2 ii stringMatching(string P){ //O(sz(P)lgn)
3     int lo=0, hi=n-1, mid=lo;
4     while(lo<hi){
5         mid=(lo+hi)/2;
6         int res=s.compare(sa[mid], sz(P), P);
7         if(res>=0) hi=mid;
8         else lo=mid+1;
9     }
10    if(s.compare(sa[lo], sz(P), P)!=0) return ii
11        (-1, -1);
12    ii ans; ans.fst=lo;
13    lo=0, hi=n-1, mid;
14    while(lo<hi){
15        mid=(lo+hi)/2;
16        int res=s.compare(sa[mid], sz(P), P);
17        if(res>0) hi=mid;
18        else lo=mid+1;
19    }
20    if(s.compare(sa[hi], sz(P), P)!=0) hi--;
21    ans.snd=hi;
22    return ans;

```

4.6. LCP (Longest Common Prefix)

```

1 //Calculates the LCP between consecutives
2 //suffixes in the Suffix Array.
3 //LCP[i] is the length of the LCP between sa[i]
4 //and sa[i-1]
5 int LCP[MAX_N], phi[MAX_N], PLCP[MAX_N];
6 void computeLCP(){//O(n)
7     phi[sa[0]]=-1;
8     forr(i, 1, n) phi[sa[i]]=sa[i-1];
9     int L=0;
10    forn(i, n){
11        if(phi[i]==-1) {PLCP[i]=0; continue;}
12        while(s[i+L]==s[phi[i]+L]) L++;
13        PLCP[i]=L;
14        L=max(L-1, 0);
15    }
16    forn(i, n) LCP[i]=PLCP[sa[i]];

```

4.7. Corasick

```

1 struct trie{
2     map<char, trie> next;
3     trie* tran[256]; //transiciones del automata
4     int idhoja, szhoja; //id de la hoja o 0 si no lo
5     es

```



```

6 //link lleva al sufijo mas largo, nxthoja lleva
  al mas largo pero que es hoja
7 trie *padre, *link, *nxthoja;
8 char pch;//caracter que conecta con padre
9 trie(): tran(), idhoja(), padre(), link() {}
10 void insert(const string &s, int id=1, int p=0)
    { //id>0!!!
11     if(p<sz(s)){
12         trie &ch=next[s[p]];
13         tran[(int)s[p]]=&ch;
14         ch.padre=this, ch.pch=s[p];
15         ch.insert(s, id, p+1);
16     }
17     else idhoja=id, szhoja=sz(s);
18 }
19 trie* get_link() {
20     if(!link){
21         if(!padre) link=this;//es la raiz
22         else if(!padre->padre) link=padre;//hijo de
           la raiz
23         else link=padre->get_link()->get_tran(pch);
24     }
25     return link; }
26 trie* get_tran(int c) {
27     if(!tran[c]) tran[c] = !padre? this : this->
        get_link()->get_tran(c);
28     return tran[c]; }
29 trie *get_nxthoja(){
30     if(!nxthoja) nxthoja = get_link()->idhoja?
        link : link->nxthoja;
31     return nxthoja; }
32 void print(int p){
33     if(idhoja) cout << "found_" << idhoja << "
        at_position_" << p-szhoja << endl;
34     if(get_nxthoja()) get_nxthoja()->print(p); }
35 void matching(const string &s, int p=0){
36     print(p); if(p<sz(s)) get_tran(s[p])->
        matching(s, p+1); }
37 }tri;
38
39
40 int main(){
41     tri=trie();//clear
42     tri.insert("ho", 1);
43     tri.insert("hoho", 2);

```

4.8. Suffix Automaton

```

1 struct state {
2     int len, link;
3     map<char,int> next;
4     state() { }
5 };
6 const int MAXLEN = 10010;
7 state st[MAXLEN*2];
8 int sz, last;
9 void sa_init() {
10     forn(i,sz) st[i].next.clear();
11     sz = last = 0;
12     st[0].len = 0;

```

```

13     st[0].link = -1;
14     ++sz;
15 }
16 // Es un DAG de una sola fuente y una sola hoja
17 // cantidad de endpos = cantidad de apariciones =
    cantidad de caminos de la clase al nodo
    terminal
18 // cantidad de miembros de la clase = st[v].len-
    st[st[v].link].len (v>0) = caminos del inicio
    a la clase
19 // El arbol de los suffix links es el suffix tree
    de la cadena invertida. La string de la
    arista link(v)->v son los caracteres que
    difieren
20 void sa_extend (char c) {
21     int cur = sz++;
22     st[cur].len = st[last].len + 1;
23     // en cur agregamos la posicion que estamos
    extendiendo
24     //podria agregar tambien un identificador de
    las cadenas a las cuales pertenece (si hay
    varias)
25     int p;
26     for (p=last; p!=-1 && !st[p].next.count(c); p=
        st[p].link) // modificar esta linea para
        hacer separadores unicos entre varias
        cadenas (c=='$')
27         st[p].next[c] = cur;
28     if (p == -1)
29         st[cur].link = 0;
30     else {
31         int q = st[p].next[c];
32         if (st[p].len + 1 == st[q].len)
33             st[cur].link = q;
34         else {
35             int clone = sz++;
36             // no le ponemos la posicion actual a clone
            sino indirectamente por el link de cur
37             st[clone].len = st[p].len + 1;
38             st[clone].next = st[q].next;
39             st[clone].link = st[q].link;
40             for (; p!=-1 && st[p].next.count(c) && st[p]
                .next[c]==q; p=st[p].link)
41                 st[p].next[c] = clone;
42             st[q].link = st[cur].link = clone;
43         }
44     }
45     last = cur;
46 }

```

4.9. Z Function

```

1 char s[MAXN];
2 int z[MAXN]; // z[i] = i==0 ? 0 : max k tq s[0,k)
    match with s[i,i+k)
3 void z_function(char s[],int z[]) {
4     int n = strlen(s);
5     forn(i, n) z[i]=0;
6     for (int i = 1, l = 0, r = 0; i < n; ++i) {
7         if (i <= r) z[i] = min (r - i + 1, z[i -

```



```

11    1]);
8    while (i + z[i] < n && s[z[i]] == s[i + z
    [i]]) ++z[i];
9    if (i + z[i] - 1 > r) l = i, r = i + z[i]
    - 1;
10   }
11  }
12
13  int main() {
14      ios::sync_with_stdio(0);

```

4.10. Palindromic tree

```

1  const int maxn = 10100100;
2
3  int len[maxn];
4  int suffLink[maxn];
5  int to[maxn][2];
6  int cnt[maxn];
7  int numV;
8  char str[maxn];
9
10 int v;
11
12 void addLetter(int n) {
13     while (str[n - len[v] - 1] != str[n] )
14         v = suffLink[v];
15     int u = suffLink[v];
16     while (str[n - len[u] - 1] != str[n] )
17         u = suffLink[u];
18     int u_ = to[u][str[n] - 'a'];
19     int v_ = to[v][str[n] - 'a'];
20     if (v_ == -1) {
21         v_ = to[v][str[n] - 'a'] = numV;
22         len[numV++] = len[v] + 2;
23         suffLink[v_] = u_;
24     }
25     v = v_;
26     cnt[v]++;
27 }
28
29 void init() {
30     memset(to, -1, sizeof to);
31     str[0] = '#'; len[0] = -1;
32     len[1] = 0; len[2] = len[3] = 1;
33     suffLink[0] = suffLink[1] = 0;
34     suffLink[2] = suffLink[3] = 1;
35     to[0][0] = 2;
36     to[0][1] = 3;
37     numV = 4;
38 }
39
40 int main() {
41     init();
42     scanf("%s", str + 1);
43     int n = strlen(str);
44     for (int i = 1; i < n; i++) addLetter(i);
45     long long ans = 0;
46     for (int i = numV - 1; i > 0; i--)
        {

```

```

47         cnt[suffLink[i] ] += cnt[i];
48         ans = max(ans, cnt[i] * 1LL * len
            [i] );
49         // fprintf(stderr, "i = %d, cnt = %d
            , len = %d\n", i, cnt[i], len[i] );
50     }
51     printf("%lld\n", ans);
52     return 0;
53 }

```

4.11. Rabin Karp - Distinct Substrings

```

1  int count_unique_substrings(string const& s) {
2      int n = s.size();
3      //ll p = 257, M = 1000000007, inv = 70038911;
4      //pow(257,10**9+7-2,10**9+7)
5      const int p = 31;
6      const int m = 1e9 + 9;
7      vector<long long> p_pow(n);
8      p_pow[0] = 1;
9      for (int i = 1; i < n; i++)
10         p_pow[i] = (p_pow[i-1] * p) % m;
11
12     vector<long long> h(n + 1, 0);
13     for (int i = 0; i < n; i++)
14         h[i+1] = (h[i] + (s[i] - 'a' + 1) * p_pow
            [i]) % m;
15
16     int cnt = 0;
17     for (int l = 1; l <= n; l++) {
18         set<long long> hs;
19         for (int i = 0; i <= n - l; i++) {
20             long long cur_h = (h[i + l] + m - h[i]
                ]) % m;
21             cur_h = (cur_h * p_pow[n-i-1]) % m;
22             hs.insert(cur_h);
23         }
24         cnt += hs.size();
25     }
26     return cnt;
27 }

```

5. Geometria

5.1. Punto

```

1  struct pto{
2      double x, y;
3      pto(double x=0, double y=0):x(x),y(y){}
4      pto operator+(pto a){return pto(x+a.x, y+a.y);}
5      pto operator-(pto a){return pto(x-a.x, y-a.y);}
6      pto operator+(double a){return pto(x+a, y+a);}
7      pto operator*(double a){return pto(x*a, y*a);}
8      pto operator/(double a){return pto(x/a, y/a);}
9      //dot product, producto interno:
10     double operator*(pto a){return x*a.x+y*a.y;}
11     //module of the cross product or vectorial
        product:
12     //if a is less than 180 clockwise from b, a^b>0
13     double operator^(pto a){return x*a.y-y*a.x;}

```

```

14 //returns true if this is at the left side of
    line qr
15 bool left(pto q, pto r){return ((q-*this)^(r-*
    this))>0;}
16 bool operator<(const pto &a) const{return x<a.x
    -EPS || (abs(x-a.x)<EPS && y<a.y-EPS);}
17 bool operator==(pto a){return abs(x-a.x)<EPS &&
    abs(y-a.y)<EPS;}
18 double norm(){return sqrt(x*x+y*y);}
19 double norm_sq(){return x*x+y*y;}
20 };
21 double dist(pto a, pto b){return (b-a).norm();}
22 typedef pto vec;
23
24 double angle(pto a, pto o, pto b){
25     pto oa=a-o, ob=b-o;
26     return atan2(oa^ob, oa*ob);}
27
28 //rotate p by theta rads CCW w.r.t. origin (0,0)
29 pto rotate(pto p, double theta){
30     return pto(p.x*cos(theta)-p.y*sin(theta),
31         p.x*sin(theta)+p.y*cos(theta));
32 }

```

5.2. Orden radial de puntos

```

1 struct Cmp{//orden total de puntos alrededor de
    un punto r
2     pto r;
3     Cmp(pto r):r(r) {}
4     int cuad(const pto &a) const{
5         if(a.x > 0 && a.y >= 0)return 0;
6         if(a.x <= 0 && a.y > 0)return 1;
7         if(a.x < 0 && a.y <= 0)return 2;
8         if(a.x >= 0 && a.y < 0)return 3;
9         assert(a.x ==0 && a.y==0);
10        return -1;
11    }
12    bool cmp(const pto&p1, const pto&p2)const{
13        int c1 = cuad(p1), c2 = cuad(p2);
14        if(c1==c2) return p1.y*p2.x<p1.x*p2.y;
15        else return c1 < c2;
16    }
17    bool operator()(const pto&p1, const pto&p2)
        const{
18        return cmp(pto(p1.x-r.x,p1.y-r.y),pto(p2.x-r.
        x,p2.y-r.y));
19    }
20 };

```

5.3. Line

```

1 int sgn(ll x){return x<0? -1 : !!x;}
2 struct line{
3     line() {}
4     double a,b,c;//Ax+By=C
5     //pto MUST store float coordinates!
6     line(double a, double b, double c):a(a),b(b),c(
        c){}
7     line(pto p, pto q): a(q.y-p.y), b(p.x-q.x), c(a
        *p.x+b*p.y) {}

```

```

8     int side(pto p){return sgn(l1(a) * p.x + l1(b)
        * p.y - c);}
9 };
10 bool parallels(line l1, line l2){return abs(l1.a*
    l2.b-l2.a*l1.b)<EPS;}
11 pto inter(line l1, line l2){//intersection
12     double det=l1.a*l2.b-l2.a*l1.b;
13     if(abs(det)<EPS) return pto(INF, INF);//
        parallels
14     return pto(l2.b*l1.c-l1.b*l2.c, l1.a*l2.c-l2.a*
        l1.c)/det;
15 }

```

5.4. Segment

```

1 struct segm{
2     pto s,f;
3     segm(pto s, pto f):s(s), f(f) {}
4     pto closest(pto p) {//use for dist to point
5         double l2 = dist_sq(s, f);
6         if(l2==0.) return s;
7         double t=((p-s)*(f-s))/l2;
8         if (t<0.) return s;//not write if is a line
9         else if(t>1.)return f;//not write if is a
            line
10        return s+((f-s)*t);
11    }
12    bool inside(pto p){return abs(dist(s, p)+dist
        (p, f)-dist(s, f))<EPS;}
13 };
14
15 pto inter(segm s1, segm s2){
16     pto r=inter(line(s1.s, s1.f), line(s2.s, s2.f))
17     ;
18     if(s1.inside(r) && s2.inside(r)) return r;
19     return pto(INF, INF);

```

5.5. Polygon Area

```

1 double area(vector<pto> &p){//0(sz(p))
2     double area=0;
3     forn(i, sz(p)) area+=p[i]^p[(i+1)%sz(p)];
4     //if points are in clockwise order then area is
        negative
5     return abs(area)/2;
6 }
7 //Area ellipse = M_PI*a*b where a and b are the
    semi axis lengths
8 //Area triangle = sqrt(s*(s-a)(s-b)(s-c)) where s
    =(a+b+c)/2

```

5.6. Circle

```

1 vec perp(vec v){return vec(-v.y, v.x);}
2 line bisector(pto x, pto y){
3     line l=line(x, y); pto m=(x+y)/2;
4     return line(-l.b, l.a, -l.b*m.x+l.a*m.y);
5 }
6 struct Circle{
7     pto o;

```

```

8   double r;
9   Circle(pto x, pto y, pto z){
10       o=inter(bisector(x, y), bisector(y, z));
11       r=dist(o, x);
12   }
13   pair<pto, pto> ptosTang(pto p){
14       pto m=(p+o)/2;
15       tipo d=dist(o, m);
16       tipo a=r*r/(2*d);
17       tipo h=sqrt(r*r-a*a);
18       pto m2=o+(m-o)*a/d;
19       vec per=perp(m-o)/d;
20       return make_pair(m2-per*h, m2+per*h);
21   }
22 };
23 //finds the center of the circle containing p1
24 //and p2 with radius r
25 //as there may be two solutions swap p1, p2 to
26 //get the other
27 bool circle2PtsRad(pto p1, pto p2, double r, pto
28 &c){
29     double d2=(p1-p2).norm_sq(), det=r*r/d2
30     -0.25;
31     if(det<0) return false;
32     c=(p1+p2)/2+perp(p2-p1)*sqrt(det);
33     return true;
34 }
35 #define sqr(a) ((a)*(a))
36 #define feq(a,b) (fabs((a)-(b))<EPS)
37 pair<tipo, tipo> ecCuad(tipo a, tipo b, tipo c){
38     //a*x*x+b*x+c=0
39     tipo dx = sqrt(b*b-4.0*a*c);
40     return make_pair((-b + dx)/(2.0*a), (-b - dx)
41         /(2.0*a));
42 }
43 pair<pto, pto> interCL(Circle c, line l){
44     bool sw=false;
45     if((sw=feq(0,l.b))){
46         swap(l.a, l.b);
47         swap(c.o.x, c.o.y);
48     }
49     pair<tipo, tipo> rc = ecCuad(
50         sqr(l.a)+sqr(l.b),
51         2.0*l.a*l.b*c.o.y-2.0*(sqr(l.b)*c.o.x+l.c*l.a),
52         sqr(l.b)*(sqr(c.o.x)+sqr(c.o.y)-sqr(c.r))+sqr(l
53             .c)-2.0*l.c*l.b*c.o.y
54     );
55     pair<pto, pto> p( pto(rc.first, (l.c - l.a * rc
56         .first) / l.b),
57         pto(rc.second, (l.c - l.a * rc.second
58             ) / l.b) );
59     if(sw){
60         swap(p.first.x, p.first.y);
61         swap(p.second.x, p.second.y);
62     }
63     return p;
64 }
65 pair<pto, pto> interCC(Circle c1, Circle c2){
66     line l;
67     l.a = c1.o.x-c2.o.x;

```

```

59     l.b = c1.o.y-c2.o.y;
60     l.c = (sqr(c2.r)-sqr(c1.r)+sqr(c1.o.x)-sqr(c2.o
61         .x)+sqr(c1.o.y)
62         -sqr(c2.o.y))/2.0;
63     return interCL(c1, l);
64 }

```

5.7. Point in Poly

```

1 //checks if v is inside of P, using ray casting
2 //works with convex and concave.
3 //excludes boundaries, handle it separately using
4 segment.inside()
5 bool inPolygon(pto v, vector<pto>& P) {
6     bool c = false;
7     forn(i, sz(P)){
8         int j=(i+1)%sz(P);
9         if((P[j].y>v.y) != (P[i].y > v.y) &&
10            (v.x < (P[i].x - P[j].x) * (v.y-P[j].y) / (P[i
11                ].y - P[j].y) + P[j].x))
12            c = !c;
13     }
14     return c;
15 }

```

5.8. Point in Convex Poly log(n)

```

1 void normalize(vector<pto> &pt){//delete
2     collinear points first!
3     //this makes it clockwise:
4     if(pt[2].left(pt[0], pt[1])) reverse(pt.begin
5         (), pt.end());
6     int n=sz(pt), pi=0;
7     forn(i, n)
8         if(pt[i].x<pt[pi].x || (pt[i].x==pt[pi].x &&
9             pt[i].y<pt[pi].y))
10             pi=i;
11     vector<pto> shift(n);//puts pi as first point
12     forn(i, n) shift[i]=pt[(pi+i)%n];
13     pt.swap(shift);
14 }
15 bool inPolygon(pto p, const vector<pto> &pt){
16     //call normalize first!
17     if(p.left(pt[0], pt[1]) || p.left(pt[sz(pt)-1],
18         pt[0])) return false;
19     int a=1, b=sz(pt)-1;
20     while(b-a>1){
21         int c=(a+b)/2;
22         if(!p.left(pt[0], pt[c])) a=c;
23         else b=c;
24     }
25     return !p.left(pt[a], pt[a+1]);
26 }

```

5.9. Convex Check CHECK

```

1 bool isConvex(vector<int> &p){//O(N), delete
2     collinear points!
3     int N=sz(p);
4     if(N<3) return false;
5     bool isLeft=p[0].left(p[1], p[2]);

```

```

5   forr(i, 1, N)
6       if(p[i].left(p[(i+1)%N], p[(i+2)%N])!=isLeft)
7           return false;
8   return true; }

```

5.10. Convex Hull

```

1 //stores convex hull of P in S, CCW order
2 //left must return >=0 to delete collinear points
3 !
4 void CH(vector<pto>& P, vector<pto> &S){
5     S.clear();
6     sort(P.begin(), P.end()); //first x, then y
7     forn(i, sz(P)){ //lower hull
8         while(sz(S)>= 2 && S[sz(S)-1].left(S[sz(S)
9             -2], P[i])) S.pop_back();
10        S.pb(P[i]);
11    }
12    S.pop_back();
13    int k=sz(S);
14    dforn(i, sz(P)){ //upper hull
15        while(sz(S) >= k+2 && S[sz(S)-1].left(S[sz(S)
16            -2], P[i])) S.pop_back();
17        S.pb(P[i]);
18    }
19    S.pop_back();
20 }

```

5.11. Cut Polygon

```

1 //cuts polygon Q along the line ab
2 //stores the left side (swap a, b for the right
3   one) in P
4 void cutPolygon(pto a, pto b, vector<pto> Q,
5     vector<pto> &P){
6     P.clear();
7     forn(i, sz(Q)){
8         double left1=(b-a)^(Q[i]-a), left2=(b-a)^(Q[
9             i+1]%sz(Q)]-a);
10        if(left1>=0) P.pb(Q[i]);
11        if(left1*left2<0)
12            P.pb(inter(line(Q[i], Q[(i+1)%sz(Q)]), line
13                (a, b)));
14    }
15 }

```

5.12. Bresenham

```

1 //plot a line approximation in a 2d map
2 void bresenham(pto a, pto b){
3     pto d=b-a; d.x=abs(d.x), d.y=abs(d.y);
4     pto s(a.x<b.x? 1: -1, a.y<b.y? 1: -1);
5     int err=d.x-d.y;
6     while(1){
7         m[a.x][a.y]=1; //plot
8         if(a==b) break;
9         int e2=err;
10        if(e2 >= 0) err-=2*d.y, a.x+=s.x;
11        if(e2 <= 0) err+= 2*d.x, a.y+= s.y;
12    }
13 }

```

5.13. Interseccion de Circulos en $n^3 \log(n)$

```

1 struct event {
2     double x; int t;
3     event(double xx, int tt) : x(xx), t(tt) {}
4     bool operator <(const event &o) const {
5         return x < o.x; }
6 };
7 typedef vector<Circle> VC;
8 typedef vector<event> VE;
9 int n;
10 double cuenta(VE &v, double A, double B) {
11     sort(v.begin(), v.end());
12     double res = 0.0, lx = ((v.empty())?0.0:v[0].
13         x);
14     int contador = 0;
15     forn(i, sz(v)) {
16         //interseccion de todos (contador == n),
17         //union de todos (contador > 0)
18         //conjunto de puntos cubierto por exacta
19         //k Circulos (contador == k)
20         if (contador == n) res += v[i].x - lx;
21         contador += v[i].t, lx = v[i].x;
22     }
23     return res;
24 }
25 // Primitiva de sqrt(r*r - x*x) como funcion
26 // de una variable x.
27 inline double primitiva(double x, double r) {
28     if (x >= r) return r*r*M_PI/4.0;
29     if (x <= -r) return -r*r*M_PI/4.0;
30     double raiz = sqrt(r*r-x*x);
31     return 0.5 * (x * raiz + r*r*atan(x/raiz));
32 }
33 double interCircle(VC &v) {
34     vector<double> p; p.reserve(v.size() * (v.
35         size() + 2));
36     forn(i, sz(v)) p.push_back(v[i].c.x + v[i].r)
37         , p.push_back(v[i].c.x - v[i].r);
38     forn(i, sz(v)) forn(j, i) {
39         Circle &a = v[i], &b = v[j];
40         double d = (a.c - b.c).norm();
41         if (fabs(a.r - b.r) < d && d < a.r + b.r)
42             {
43                 double alfa = acos((sqr(a.r) + sqr(d)
44                     - sqr(b.r)) / (2.0 * d * a.r));
45                 pto vec = (b.c - a.c) * (a.r / d);
46                 p.pb((a.c + rotate(vec, alfa)).x), p.
47                     pb((a.c + rotate(vec, -alfa)).x);
48             }
49     }
50     sort(p.begin(), p.end());
51     double res = 0.0;
52     forn(i, sz(p)-1) {
53         const double A = p[i], B = p[i+1];
54         VE ve; ve.reserve(2 * v.size());
55         forn(j, sz(v)) {
56             const Circle &c = v[j];
57             double arco = primitiva(B-c.c.x, c.r)
58                 - primitiva(A-c.c.x, c.r);

```

```

48         double base = c.c.y * (B-A);
49         ve.push_back(event(base + arco,-1));
50         ve.push_back(event(base - arco, 1));
51     }
52     res += cuenta(ve,A,B);
53 }
54 return res;
55 }

```

6. Math

6.1. Identidades

$\sum_{i=0}^n i \binom{n}{i} = n * 2^{n-1}$
 $\binom{n}{k} = \frac{n}{k} \binom{n-1}{k-1}$
 $\sum_{i=0}^k \binom{a}{0} \binom{b}{k-i} = \binom{a+b}{k}$
 $\sum_{i=0}^m \binom{n+i}{i} = \binom{n+m+1}{n+1}$
 $\sum_{i=0}^n i(i-1) = \frac{8}{6} \left(\frac{n}{2}\right) \left(\frac{n}{2}+1\right) (n+1)$ (doubles) \rightarrow Sino ver caso impar y par

$$\sum_{i=0}^n i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} = \frac{n^5}{5} + \frac{n^4}{2} + \frac{n^3}{3} - \frac{n}{30}$$

$$\sum_{i=0}^n i^p = \frac{(n+1)^{p+1}}{p+1} + \sum_{k=1}^p \frac{B_k}{p-k+1} \binom{p}{k} (n+1)^{p-k+1}$$

$$\sum_{i=0}^n i^p = \frac{(n+1)^{p+1}}{p+1} + \sum_{k=1}^p \frac{B_k}{p-k+1} \binom{p}{k} (n+1)^{p-k+1}$$

Caras + Vertices = Aristas + 2 (Poliedros convexos y Grafos planos)

Teorema de Pick: (Area, puntos int. y puntos en el borde)

$$A = I + \frac{B}{2} - 1$$

$\left\{ \begin{smallmatrix} n+1 \\ k \end{smallmatrix} \right\} = k \left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\} + \left\{ \begin{smallmatrix} n \\ k-1 \end{smallmatrix} \right\}$ for $k > 0$ with initial conditions
 $\left\{ \begin{smallmatrix} 0 \\ 0 \end{smallmatrix} \right\} = 1$ and $\left\{ \begin{smallmatrix} n \\ 0 \end{smallmatrix} \right\} = \left\{ \begin{smallmatrix} 0 \\ n \end{smallmatrix} \right\} = 0$ for $n > 0$. Same as
 $\frac{1}{k!} \sum_{j=0}^k (-1)^{k-j} \binom{k}{j} j^n$

$\left[\begin{smallmatrix} n+1 \\ k \end{smallmatrix} \right] = n \left[\begin{smallmatrix} n \\ k \end{smallmatrix} \right] + \left[\begin{smallmatrix} n \\ k-1 \end{smallmatrix} \right]$ for $k > 0$, with the initial conditions
 $\left[\begin{smallmatrix} 0 \\ 0 \end{smallmatrix} \right] = 1$ and $\left[\begin{smallmatrix} n \\ 0 \end{smallmatrix} \right] = \left[\begin{smallmatrix} 0 \\ n \end{smallmatrix} \right] = 0$ for $n > 0$.

Markov Chain: $\begin{pmatrix} Q & R \\ 0 & I_r \end{pmatrix}$ Q transient st., R absorb. st.

Fundamental Matrix: $N = \sum_k Q^k = (I_t - Q)^{-1}$.

Steps before absorption: $t = N \cdot 1$

Absorption probability: $B = N \cdot R$

Prob. visiting transient state: $H = (N - I_t) N_{dg}^{-1}$

Usage Example: GaussJordan((I-Q),b), where b is a dummy column of 0's. Each row i is as if they started at state i .

6.2. Ec. Caracteristica

$$a_0 T(n) + a_1 T(n-1) + \dots + a_k T(n-k) = 0$$

$$p(x) = a_0 x^k + a_1 x^{k-1} + \dots + a_k$$

Sean r_1, r_2, \dots, r_q las raíces distintas, de mult. m_1, m_2, \dots, m_q

$$T(n) = \sum_{i=1}^q \sum_{j=0}^{m_i-1} c_{ij} n^j r_i^n$$

Las constantes c_{ij} se determinan por los casos base.

6.3. Combinatorio

```

1  forn(i, MAXN+1){ //comb[i][k]=i tomados de a k
2      comb[i][0]=comb[i][i]=1;
3      forr(k, 1, i) comb[i][k]=(comb[i-1][k]+comb[i-1][k-1])%MOD;
4  }
5  ll lucas (ll n, ll k, int p){ //Calcula (n,k)%p
    teniendo comb[p][p] precalculado.

```

```

6  ll aux = 1;
7  while (n + k) aux = (aux * comb[n%p][k%p]) %p,
    n/=p, k/=p;
8  return aux;
9  }

```

6.4. Gauss Jordan, Determinante

```

1  // Gauss-Jordan elimination with full pivoting.
2  //
3  // Uses:
4  // (1) solving systems of linear equations (AX=B)
5  // (2) inverting matrices (AX=I)
6  // (3) computing determinants of square matrices
7  //
8  // Running time: O(n^3)
9  //
10 // INPUT:      a[] [] = an nxn matrix
11 //              b[] [] = an nxm matrix
12 //
13 // OUTPUT:
14 // X= an nxm matrix (stored in b[] [])
15 // A^{-1} = an nxn matrix (stored in a[] [])
16 // returns determinant of a[] []
17
18 #include <iostream>
19 #include <vector>
20 #include <cmath>
21
22 using namespace std;
23
24 const double EPS = 1e-10;
25
26 typedef vector<int> VI;
27 typedef double T;
28 typedef vector<T> VT;
29 typedef vector<VT> VVT;
30
31 T GaussJordan(VVT &a, VVT &b) {
32     const int n = a.size();
33     const int m = b[0].size();
34     VI irow(n), icol(n), ipiv(n);
35     T det = 1;
36
37     for (int i = 0; i < n; i++) {
38         int pj = -1, pk = -1;
39         for (int j = 0; j < n; j++) if (!ipiv[j])
40             for (int k = 0; k < n; k++) if (!ipiv[k])
41                 if (pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk]))
42                     { pj = j; pk = k; }
43         if (fabs(a[pj][pk]) < EPS) { cerr << "Matrix_
44             is_singular." << endl; exit(0); }
45         ipiv[pk]++;
46         swap(a[pj], a[pk]);
47         swap(b[pj], b[pk]);
48         if (pj != pk) det *= -1;
49         irow[i] = pj;
50         icol[i] = pk;
51
52         T c = 1.0 / a[pk][pk];

```

```

51     det *= a[pk][pk];
52     a[pk][pk] = 1.0;
53     for (int p = 0; p < n; p++) a[pk][p] *= c;
54     for (int p = 0; p < m; p++) b[pk][p] *= c;
55     for (int p = 0; p < n; p++) if (p != pk) {
56         c = a[p][pk];
57         a[p][pk] = 0;
58         for (int q = 0; q < n; q++) a[p][q] -= a[pk][q] * c;
59         for (int q = 0; q < m; q++) b[p][q] -= b[pk][q] * c;
60     }
61 }
62
63 for (int p = n-1; p >= 0; p--) if (irow[p] !=
64     icol[p]) {
65     for (int k = 0; k < n; k++) swap(a[k][irow[p]
66         ], a[k][icol[p]]);
67 }
68
69 return det;
70 }
71
72 int main() {
73     const int n = 4;
74     const int m = 2;
75     double A[n][n] = {
76         {1,2,3,4},{1,0,1,0},{5,3,2,4},{6,1,4,6} };
77     double B[n][m] = { {1,2},{4,3},{5,6},{8,7} };
78     VVT a(n), b(m);
79     for (int i = 0; i < n; i++) {
80         a[i] = VT(A[i], A[i] + n);
81         b[i] = VT(B[i], B[i] + m);
82     }
83
84     double det = GaussJordan(a, b);
85
86     // expected: 60
87     cout << "Determinant:␣" << det << endl;
88
89     // expected:
90     // -0.233333 0.166667 0.133333 0.066667
91     // 0.166667 0.166667 0.333333 -0.333333
92     // 0.233333 0.833333 -0.133333 -0.066667
93     // 0.05 -0.75 -0.1 0.2
94     cout << "Inverse:␣" << endl;
95     for (int i = 0; i < n; i++) {
96         for (int j = 0; j < n; j++)
97             cout << a[i][j] << '␣';
98         cout << endl;
99     }
100
101     // expected: 1.63333 1.3
102     // -0.166667 0.5
103     // 2.36667 1.7
104     // -1.85 -1.35
105     cout << "Solution:␣" << endl;
106     for (int i = 0; i < n; i++) {
107         for (int j = 0; j < m; j++)
108             cout << b[i][j] << '␣';

```

```

106     cout << endl;
107 }
108 }

```

6.5. Teorema Chino del Resto

```

1 // Chinese remainder theorem (special case): find
2 // z such that
3 // z % m1 = r1, z % m2 = r2. Here, z is unique
4 // modulo M = lcm(m1, m2).
5 // Return (z, M). On failure, M = -1.
6 PII chinese_remainder_theorem(int m1, int r1, int
7     m2, int r2) {
8     int s, t;
9     int g = extended_euclid(m1, m2, s, t);
10    if (r1%g != r2%g) return make_pair(0, -1);
11    return make_pair(mod(s*r2*m1 + t*r1*m2, m1*m2)
12        / g, m1*m2 / g);
13 }
14
15 // Chinese remainder theorem: find z such that
16 // z % m[i] = r[i] for all i. Note that the
17 // solution is
18 // unique modulo M = lcm_i (m[i]). Return (z, M)
19 // . On
20 // failure, M = -1. Note that we do not require
21 // the a[i]'s
22 // to be relatively prime.
23 PII chinese_remainder_theorem(const VI &m, const
24     VI &r) {
25     PII ret = make_pair(r[0], m[0]);
26     for (int i = 1; i < m.size(); i++) {
27         ret = chinese_remainder_theorem(ret.second,
28             ret.first, m[i], r[i]);
29         if (ret.second == -1) break;
30     }
31     return ret;
32 }

```

6.6. Funciones de primos

Iterar mientras el $p^2 \leq N$. Revisar que $N! = 1$, en este caso N es primo. **NumDiv**: Producto (exponentes+1). **SumDiv**: Product suma geom. factores. **EulerPhi** (coprimos): Inicia $ans = N$. Para cada primo divisor: $ans = ans / \text{primo}$ (una vez) y dividir luego N todo lo posible por p .

6.7. Phollard's Rho (rolando)

```

1 ll gcd(ll a, ll b){return a?gcd(b %a, a):b;}
2
3 ll mulmod (ll a, ll b, ll c) { //returns (a*b)%c,
4     //and minimize overflow
5     ll x = 0, y = a%c;
6     while (b > 0){
7         if (b % 2 == 1) x = (x+y) % c;
8         y = (y*2) % c;
9         b /= 2;
10    }
11    return x % c;

```



```

12 ll expmod (ll b, ll e, ll m){//O(log b)
13     if(!e) return 1;
14     ll q= expmod(b,e/2,m); q=mulmod(q,q,m);
15     return e%2? mulmod(b,q,m) : q;
16 }
17
18
19 bool es_primo_prob (ll n, int a)
20 {
21     if (n == a) return true;
22     ll s = 0,d = n-1;
23     while (d % 2 == 0) s++,d/=2;
24
25     ll x = expmod(a,d,n);
26     if ((x == 1) || (x+1 == n)) return true;
27
28     forn (i, s-1){
29         x = mulmod(x, x, n);
30         if (x == 1) return false;
31         if (x+1 == n) return true;
32     }
33     return false;
34 }
35
36 bool rabin (ll n){ //devuelve true si n es primo
37     if (n == 1) return false;
38     const int ar[] = {2,3,5,7,11,13,17,19,23};
39     forn (j,9)
40         if (!es_primo_prob(n,ar[j]))
41             return false;
42     return true;
43 }
44
45 ll rho(ll n){
46     if( (n & 1) == 0 ) return 2;
47     ll x = 2 , y = 2 , d = 1;
48     ll c = rand() % n + 1;
49     while( d == 1 ){
50         x = (mulmod( x , x , n ) + c)%n;
51         y = (mulmod( y , y , n ) + c)%n;
52         y = (mulmod( y , y , n ) + c)%n;
53         if( x - y >= 0 ) d = gcd( x - y , n );
54         else d = gcd( y - x , n );
55     }
56     return d==n? rho(n):d;
57 }
58
59 map<ll,ll> prim;
60 void factRho (ll n){ //O (lg n)^3. un solo numero
61     if (n == 1) return;
62     if (rabin(n)){
63         prim[n]++;
64         return;
65     }
66     ll factor = rho(n);
67     factRho(factor);
68     factRho(n/factor);
69 }

```

6.8. GCD

```

1 tipo gcd(tipo a, tipo b){return a?gcd(b %a, a):b
   ;}

```

6.9. Extended Euclid

```

1 void extendedEuclid (ll a, ll b){ //a * x + b * y
   = d
2     if (!b) { x = 1; y = 0; d = a; return;}
3     extendedEuclid (b, a%b);
4     ll x1 = y;
5     ll y1 = x - (a/b) * y;
6     x = x1; y = y1;
7 }

```

6.10. Polinomio

```

1     int m = sz(c), n = sz(o.c);
2     vector<tipo> res(max(m,n));
3     forn(i, m) res[i] += c[i];
4     forn(i, n) res[i] += o.c[i];
5     return poly(res);    }
6 poly operator*(const tipo cons) const {
7     vector<tipo> res(sz(c));
8     forn(i, sz(c)) res[i]=c[i]*cons;
9     return poly(res);    }
10 poly operator*(const poly &o) const {
11     int m = sz(c), n = sz(o.c);
12     vector<tipo> res(m+n-1);
13     forn(i, m) forn(j, n) res[i+j]+=c[i]*o.c[
        j];
14     return poly(res);    }
15 tipo eval(tipo v) {
16     tipo sum = 0;
17     dforn(i, sz(c)) sum=sum*v + c[i];
18     return sum; }
19 //poly contains only a vector<int> c (the
   coeficients)
20 //the following function generates the roots of
   the polynomial
21 //it can be easily modified to return float roots
22 set<tipo> roots(){
23     set<tipo> roots;
24     tipo a0 = abs(c[0]), an = abs(c[sz(c)-1]);
25     vector<tipo> ps,qs;
26     forr(p,1,sqrt(a0)+1) if (a0%p==0) ps.pb(p),ps
        .pb(a0/p);
27     forr(q,1,sqrt(an)+1) if (an%q==0) qs.pb(q),qs
        .pb(an/q);
28     forall(pt,ps)
29         forall(qt,qs) if ( (*pt) % (*qt)==0 ) {
30             tipo root = abs((*pt) / (*qt));
31             if (eval(root)==0) roots.insert(root);
32         }
33     return roots; }
34 };
35 pair<poly,tipo> ruffini(const poly p, tipo r) {
36     int n = sz(p.c) - 1 ;
37     vector<tipo> b(n);
38     b[n-1] = p.c[n];
39     dforn(k,n-1) b[k] = p.c[k+1] + r*b[k+1];
40     tipo resto = p.c[0] + r*b[0];

```

```

41 poly result(b);
42 return make_pair(result, resto);
43 }
44 poly interpolate(const vector<tipo>& x, const
    vector<tipo>& y) {
45     poly A; A.c.pb(1);
46     forn(i, sz(x)) { poly aux; aux.c.pb(-x[i]),
        aux.c.pb(1), A = A * aux; }
47     poly S; S.c.pb(0);
48     forn(i, sz(x)) { poly Li;
49         Li = ruffini(A, x[i]).fst;
50         Li = Li * (1.0 / Li.eval(x[i])); // here put
            a multiple of the coefficients instead of
            1.0 to avoid using double
51         S = S + Li * y[i]; }
52     return S;
53 }
54
55 int main(){
56     return 0;
57 }

```

6.11. FFT

```

1 //~ typedef complex<double> base; //menos codigo,
    pero mas lento
2 //elegir si usar complejos de c (lento) o estos
3 struct base{
4     double r, i;
5     base(double r=0, double i=0):r(r), i(i){}
6     double real()const{return r;}
7     void operator/=(const int c){r/=c, i/=c;}
8 };
9 base operator*(const base &a, const base &b){
10     return base(a.r*b.r-a.i*b.i, a.r*b.i+a.i*b.r)
        ;}
11 base operator+(const base &a, const base &b){
12     return base(a.r+b.r, a.i+b.i);}
13 base operator-(const base &a, const base &b){
14     return base(a.r-b.r, a.i-b.i);}
15 vector<int> rev; vector<base> wlen_pw;
16 inline static void fft(base a[], int n, bool
    invert) {
17     forn(i, n) if(i<rev[i]) swap(a[i], a[rev[i]])
        ;
18     for (int len=2; len<=n; len<=1) {
19         double ang = 2*M_PI/len * (invert?-1:+1);
20         int len2 = len>>1;
21         base wlen (cos(ang), sin(ang));
22         wlen_pw[0] = base (1, 0);
23         forr(i, 1, len2) wlen_pw[i] = wlen_pw[i
            -1] * wlen;
24         for (int i=0; i<n; i+=len) {
25             base t, *pu = a+i, *pv = a+i+len2, *pu_end
                = a+i+len2, *pw = &wlen_pw[0];
26             for (; pu!=pu_end; ++pu, ++pv, ++pw)
27                 t = *pv * *pw, *pv = *pu - t, *pu = *pu +
                    t;
28         }
29 }

```

```

30 if (invert) forn(i, n) a[i]/= n;}
31 inline static void calc_rev(int n){//precalculo:
    llamar antes de fft!!
32     wlen_pw.resize(n), rev.resize(n);
33     int lg=31-__builtin_clz(n);
34     forn(i, n){
35         rev[i] = 0;
36         forn(k, lg) if(i&(1<<k)) rev[i]|=1<<(lg
            -1-k);
37     }}
38 inline static void multiply(const vector<int> &a,
    const vector<int> &b, vector<int> &res) {
39     vector<base> fa (a.begin(), a.end()), fb (b.
        begin(), b.end());
40     int n=1; while(n < max(sz(a), sz(b))) n <<=
        1; n <<= 1;
41     calc_rev(n);
42     fa.resize (n), fb.resize (n);
43     fft (&fa[0], n, false), fft (&fb[0], n, false)
        ;
44     forn(i, n) fa[i] = fa[i] * fb[i];
45     fft (&fa[0], n, true);
46     res.resize(n);
47     forn(i, n) res[i] = int (fa[i].real() + 0.5);
        }
48 void toPoly(const string &s, vector<int> &P){//
    convierte un numero a polinomio
49     P.clear();
50     dforn(i, sz(s)) P.pb(s[i]-'0');}

```

7. Grafos

7.1. Bellman-Ford

```

1 vector<ii> G[MAX_N]; //ady. list with pairs (
    weight, dst)
2 int dist[MAX_N];
3 void bford(int src){//O(VE)
4     dist[src]=0;
5     forn(i, N-1) forn(j, N) if(dist[j]!=INF) forall
        (it, G[j])
6         dist[it->snd]=min(dist[it->snd], dist[j]+it->
            fst);
7 }
8
9 bool hasNegCycle(){
10     forn(j, N) if(dist[j]!=INF) forall(it, G[j])
11         if(dist[it->snd]>dist[j]+it->fst) return true
            ;
12     //inside if: all points reachable from it->snd
        will have -INF distance(do bfs)
13     return false;
14 }

```

7.2. 2-SAT + Tarjan SCC

```

1 //We have a vertex representing a var and other
    for his negation.
2 //Every edge stored in G represents an
    implication. To add an equation of the form a
    ||b, use addor(a, b)

```

```

3 //MAX=max cant var, n=cant var
4 #define addor(a, b) (G[neg(a)].pb(b), G[neg(b)].
    pb(a))
5 vector<int> G[MAX*2];
6 //idx[i]=index assigned in the dfs
7 //lw[i]=lowest index(closer from the root)
    reachable from i
8 int lw[MAX*2], idx[MAX*2], qidx;
9 stack<int> q;
10 int qcmp, cmp[MAX*2];
11 //verdad[cmp[i]]=valor de la variable i
12 bool verdad[MAX*2+1];
13
14 int neg(int x) { return x>=n? x-n : x+n;}
15 void tjn(int v){
16     lw[v]=idx[v]=++qidx;
17     q.push(v), cmp[v]=-2;
18     forall(it, G[v]){
19         if(!idx[*it] || cmp[*it]==-2){
20             if(!idx[*it]) tjn(*it);
21             lw[v]=min(lw[v], lw[*it]);
22         }
23     }
24     if(lw[v]==idx[v]){
25         int x;
26         do{x=q.top(); q.pop(); cmp[x]=qcmp;}while(x!=
            v);
27         verdad[qcmp]=(cmp[neg(v)]<0);
28         qcmp++;
29     }
30 }
31 //remember to CLEAR G!!!
32 bool satisf(){//O(n)
33     memset(idx, 0, sizeof(idx)), qidx=0;
34     memset(cmp, -1, sizeof(cmp)), qcmp=0;
35     forn(i, n){
36         if(!idx[i]) tjn(i);
37         if(!idx[neg(i)]) tjn(neg(i));
38     }
39     forn(i, n) if(cmp[i]==cmp[neg(i)]) return false
        ;
40     return true;
41 }

```

7.3. Puentes y Articulation Points

```

1 int dfsNumberCounter, dfsRoot, rootChildren;
2 vi dfs_num, dfs_low, dfs_parent,
    articulation_vertex;
3
4 void articulationPointAndBridge(int u) {
5     dfs_low[u] = dfs_num[u] = dfsNumberCounter++;
6     for (int j = 0; j < (int)AdjList[u].size(); j
        ++){
7         ii v = AdjList[u][j];
8         if (dfs_num[v.first] == -1) {
9             dfs_parent[v.first] = u;
10            if (u == dfsRoot) rootChildren++;
11            articulationPointAndBridge(v.first);
12            if (dfs_low[v.first] >= dfs_num[u])

```

```

13         articulation_vertex[u] = true;
14         if (dfs_low[v.first] > dfs_num[u])
15             printf("Edge (%d,%d) is a bridge\n", u,
                v.first);
16         dfs_low[u] = min(dfs_low[u], dfs_low[v.
            first]);
17     }
18     else if (v.first != dfs_parent[u])
19         dfs_low[u] = min(dfs_low[u], dfs_num[v.
            first]);
20 } }
21 // At main
22 dfsNumberCounter = 0; dfs_num.assign(V, -1);
23 dfs_low.assign(V, 0);
24 dfs_parent.assign(V, -1); articulation_vertex.
    assign(V, 0);
25 printf("Bridges:\n");
26 for (int i = 0; i < V; i++)
27     if (dfs_num[i] == -1) {
28         dfsRoot = i; rootChildren = 0;
29         articulationPointAndBridge(i);
30         articulation_vertex[dfsRoot] = (
            rootChildren > 1); }
31 printf("Articulation Points:\n");
32 for (int i = 0; i < V; i++)
33     if (articulation_vertex[i])
34         printf("Vertex %d\n", i);

```

7.4. LCA + Climb

```

1 const int MAXN=100001;
2 const int LOGN=20;
3 //f[v][k] holds the 2^k father of v
4 //L[v] holds the level of v
5 int N, f[MAXN][LOGN], L[MAXN];
6 //call before build:
7 void dfs(int v, int fa=-1, int lvl=0){//generate
    required data
8     f[v][0]=fa, L[v]=lvl;
9     forall(it, G[v])if(*it!=fa) dfs(*it, v, lvl+1);
10 }
11 void build(){//f[i][0] must be filled previously,
    O(nlgn)
12     forn(k, LOGN-1) forn(i, N) f[i][k+1]=f[f[i][k]
        ][k];}
13 #define lg(x) (31-__builtin_clz(x))//=floor(log2(
    x))
14 int climb(int a, int d){//O(lgn)
15     if(!d) return a;
16     dforn(i, lg(L[a])+1) if(1<=<i<=d) a=f[a][i], d
        -=1<=<i;
17     return a;}
18 int lca(int a, int b){//O(lgn)
19     if(L[a]<L[b]) swap(a, b);
20     a=climb(a, L[a]-L[b]);
21     if(a==b) return a;
22     dforn(i, lg(L[a])+1) if(f[a][i]!=f[b][i]) a=f[a
        ][i], b=f[b][i];
23     return f[a][0]; }
24 int dist(int a, int b) {//returns distance

```

```

    between nodes
24 | return L[a]+L[b]-2*L[lca(a, b)];}

```

7.5. Heavy Light Decomposition

```

1 | int treesz[MAXN]; //cantidad de nodos en el
    subarbol del nodo v
2 | int dad[MAXN]; //dad[v]=padre del nodo v
3 | void dfs1(int v, int p=-1){ //pre-dfs
4 |     dad[v]=p;
5 |     treesz[v]=1;
6 |     forall(it, G[v]) if(*it!=p){
7 |         dfs1(*it, v);
8 |         treesz[v]+=treesz[*it];
9 |     }
10 | }
11 | //PONER Q EN 0 !!!!!
12 | int pos[MAXN], q; //pos[v]=posicion del nodo v en
    el recorrido de la dfs
13 | //Las cadenas aparecen continuas en el recorrido!
14 | int cantcad;
15 | int homecad[MAXN]; //dada una cadena devuelve su
    nodo inicial
16 | int cad[MAXN]; //cad[v]=cadena a la que pertenece
    el nodo
17 | void heavylight(int v, int cur=-1){
18 |     if(cur==-1) homecad[cur=cantcad++]=v;
19 |     pos[v]=q++;
20 |     cad[v]=cur;
21 |     int mx=-1;
22 |     forn(i, sz(G[v])) if(G[v][i]!=dad[v])
23 |         if(mx==-1 || treesz[G[v][mx]]<treesz[G[v][i]]
24 |             || mx==i) mx=i;
25 |     if(mx!=-1) heavylight(G[v][mx], cur);
26 |     forn(i, sz(G[v])) if(i!=mx && G[v][i]!=dad[v])
27 |         heavylight(G[v][i], -1);
28 | }
29 | //ejemplo de obtener el maximo numero en el
    camino entre dos nodos
30 | //RTA: max(query(low, u), query(low, v)), con low
    =lca(u, v)
31 | //esta funcion va trepando por las cadenas
32 | int query(int an, int v){ //O(logn)
33 |     //si estan en la misma cadena:
34 |     if(cad[an]==cad[v]) return rmq.get(pos[an], pos
35 |         [v]+1);
36 |     return max(query(an, dad[homecad[cad[v]]]),
37 |         rmq.get(pos[homecad[cad[v]]], pos[v]+1))
38 |     ;
39 | }

```

7.6. Centroid Decomposition

```

1 | int n;
2 | vector<int> G[MAXN];
3 | bool taken[MAXN]; //poner todos en FALSE al
    principio!!
4 | int padre[MAXN]; //padre de cada nodo en el
    centroid tree
5 |
6 | int szt[MAXN];

```

```

7 | void calcsz(int v, int p) {
8 |     szt[v] = 1;
9 |     forall(it, G[v]) if (*it!=p && !taken[*it])
10 |         calcsz(*it, v), szt[v]+=szt[*it];
11 | }
12 | void centroid(int v=0, int f=-1, int lvl=0, int
    tam=-1) { //O(nlogn)
13 |     if(tam==-1) calcsz(v, -1), tam=szt[v];
14 |     forall(it, G[v]) if(!taken[*it] && szt[*it]>=
15 |         tam/2)
16 |         {szt[v]=0; centroid(*it, f, lvl, tam); return
17 |             ;}
18 |     taken[v]=true;
19 |     padre[v]=f;
20 |     forall(it, G[v]) if(!taken[*it])
21 |         centroid(*it, v, lvl+1, -1);
22 | }

```

7.7. Euler Cycle

```

1 | int n,m,ars[MAXE], eq;
2 | vector<int> G[MAXN]; //fill G,n,m,ars,eq
3 | list<int> path;
4 | int used[MAXN];
5 | bool usede[MAXE];
6 | queue<list<int>::iterator> q;
7 | int get(int v){
8 |     while(used[v]<sz(G[v]) && usede[ G[v][used[v]]
9 |         ]) used[v]++;
10 |     return used[v];
11 | }
12 | void explore(int v, int r, list<int>::iterator it
13 |     ){
14 |     int ar=G[v][get(v)]; int u=v^ars[ar];
15 |     usede[ar]=true;
16 |     list<int>::iterator it2=path.insert(it, u);
17 |     if(u!=r) explore(u, r, it2);
18 |     if(get(v)<sz(G[v])) q.push(it);
19 | }
20 | void euler(){
21 |     zero(used), zero(usede);
22 |     path.clear();
23 |     q=queue<list<int>::iterator>();
24 |     path.push_back(0); q.push(path.begin());
25 |     while(sz(q)){
26 |         list<int>::iterator it=q.front(); q.pop();
27 |         if(used[*it]<sz(G[*it])) explore(*it, *it, it
28 |             );
29 |     }
30 |     reverse(path.begin(), path.end());
31 | }
32 | void addEdge(int u, int v){
33 |     G[u].pb(eq), G[v].pb(eq);
34 |     ars[eq++]=u^v;
35 | }

```

7.8. Chu-liu

```

1 | void visit(graph &h, int v, int s, int r,
2 |     vector<int> &no, vector< vector<int> > &comp,

```

```

3  vector<int> &prev, vector< vector<int> > &next,
   vector<weight> &mcost,
4  vector<int> &mark, weight &cost, bool &found) {
5  if (mark[v]) {
6      vector<int> temp = no;
7      found = true;
8      do {
9          cost += mcost[v];
10         v = prev[v];
11         if (v != s) {
12             while (comp[v].size() > 0) {
13                 no[comp[v].back()] = s;
14                 comp[s].push_back(comp[v].back());
15                 comp[v].pop_back();
16             }
17         }
18     } while (v != s);
19     forall(j, comp[s]) if (*j != r) forall(e, h[*j
20         ] )
21         if (no[e->src] != s) e->w -= mcost[ temp[*j
22             ] ];
23     mark[v] = true;
24     forall(i, next[v]) if (no[*i] != no[v] && prev[
25         no[*i]] == v)
26         if (!mark[no[*i]] || *i == s)
27             visit(h, *i, s, r, no, comp, prev, next,
28                 mcost, mark, cost, found);
29 }
30 weight minimumSpanningArborescence(const graph &g
31     , int r) {
32     const int n=sz(g);
33     graph h(n);
34     forn(u,n) forall(e,g[u]) h[e->dst].pb(*e);
35     vector<int> no(n);
36     vector<vector<int> > comp(n);
37     forn(u, n) comp[u].pb(no[u] = u);
38     for (weight cost = 0; ; ) {
39         vector<int> prev(n, -1);
40         vector<weight> mcost(n, INF);
41         forn(j,n) if (j != r) forall(e,h[j])
42             if (no[e->src] != no[j])
43                 if (e->w < mcost[ no[j] ])
44                     mcost[ no[j] ] = e->w, prev[ no[j] ] =
45                         no[e->src];
46         vector< vector<int> > next(n);
47         forn(u,n) if (prev[u] >= 0)
48             next[ prev[u] ].push_back(u);
49         bool stop = true;
50         vector<int> mark(n);
51         forn(u,n) if (u != r && !mark[u] && !comp[u].
52             empty()) {
53             bool found = false;
54             visit(h, u, u, r, no, comp, prev, next,
55                 mcost, mark, cost, found);
56             if (found) stop = false;
57         }
58     }
59     if (stop) {
60         forn(u,n) if (prev[u] >= 0) cost += mcost[u
61             ];
62     }

```

```

53     return cost;
54 }
55 }
56 }

```

7.9. Hungarian

```

1  //Dado un grafo bipartito completo con costos no
   negativos, encuentra el matching perfecto de
   minimo costo.
2  tipo cost[N][N], lx[N], ly[N], slack[N]; //llenar
   : cost=matriz de adyacencia
3  int n, max_match, xy[N], yx[N], slackx[N], prev2[N
4  ]; //n=cantidad de nodos
5  bool S[N], T[N]; //sets S and T in algorithm
6  void add_to_tree(int x, int prevx) {
7      S[x] = true, prev2[x] = prevx;
8      forn(y, n) if (lx[x] + ly[y] - cost[x][y] <
9          slack[y] - EPS)
10         slack[y] = lx[x] + ly[y] - cost[x][y], slackx
11             [y] = x;
12 }
13 void update_labels(){
14     tipo delta = INF;
15     forn (y, n) if (!T[y]) delta = min(delta, slack
16         [y]);
17     forn (x, n) if (S[x]) lx[x] -= delta;
18     forn (y, n) if (T[y]) ly[y] += delta; else
19         slack[y] -= delta;
20 }
21 void init_labels(){
22     zero(lx), zero(ly);
23     forn (x,n) forn(y,n) lx[x] = max(lx[x], cost[x
24         ][y]);
25 }
26 void augment() {
27     if (max_match == n) return;
28     int x, y, root, q[N], wr = 0, rd = 0;
29     memset(S, false, sizeof(S)), memset(T, false,
30         sizeof(T));
31     memset(prev2, -1, sizeof(prev2));
32     forn (x, n) if (xy[x] == -1){
33         q[wr++] = root = x, prev2[x] = -2;
34         S[x] = true; break; }
35     forn (y, n) slack[y] = lx[root] + ly[y] - cost[
36         root][y], slackx[y] = root;
37     while (true){
38         while (rd < wr){
39             x = q[rd++];
40             for (y = 0; y < n; y++) if (cost[x][y] ==
41                 lx[x] + ly[y] && !T[y]){
42                 if (yx[y] == -1) break; T[y] = true;
43                 q[wr++] = yx[y], add_to_tree(yx[y], x); }
44             if (y < n) break; }
45         if (y < n) break;
46         update_labels(), wr = rd = 0;
47         forn (y = 0; y < n; y++) if (!T[y] && slack[y]
48             == 0){
49             if (yx[y] == -1){x = slackx[y]; break;}
50             else{

```



```

41     T[y] = true;
42     if (!S[yx[y]]) q[wr++] = yx[y],
        add_to_tree(yx[y], slackx[y]);
43 }
44 if (y < n) break; }
45 if (y < n){
46     max_match++;
47     for (int cx = x, cy = y, ty; cx != -2; cx =
        prev2[cx], cy = ty)
48         ty = xy[cx], yx[cy] = cx, xy[cx] = cy;
49     augment(); }
50 }
51 tipo hungarian(){
52     tipo ret = 0; max_match = 0, memset(xy, -1,
        sizeof(xy));
53     memset(yx, -1, sizeof(yx)), init_labels(),
        augment(); //steps 1-3
54     forn (x,n) ret += cost[x][xy[x]]; return ret;
55 }

```

7.10. Dynamic Connectivity

```

1 struct UnionFind {
2     int n, comp;
3     vector<int> pre, si, c;
4     UnionFind(int n=0):n(n), comp(n), pre(n), si(
        n, 1) {
5         forn(i,n) pre[i] = i; }
6     int find(int u){return u==pre[u]?u:find(pre[u]
        )};
7     bool merge(int u, int v) {
8         if((u=find(u))==v) return false
9         ;
10        if(si[u]<si[v]) swap(u, v);
11        si[u]+=si[v], pre[v]=u, comp--, c.pb(v);
12        return true;
13    }
14    int snap(){return sz(c);}
15    void rollback(int snap){
16        while(sz(c)>snap){
17            int v = c.back(); c.pop_back();
18            si[pre[v]] -= si[v], pre[v] = v, comp
19            ++;
20        }
21    };
22    enum {ADD,DEL,QUERY};
23    struct Query {int type,u,v;};
24    struct DynCon {
25        vector<Query> q;
26        UnionFind dsu;
27        vector<int> match,res;
28        map<ii,int> last;//se puede no usar cuando
29        hay identificador para cada arista (
30        mejora poco)
31        DynCon(int n=0):dsu(n){}
32        void add(int u, int v) {
33            if(u>v) swap(u,v);
34            q.pb((Query){ADD, u, v}), match.pb(-1);
35            last[ii(u,v)] = sz(q)-1;

```

```

33    }
34    void remove(int u, int v) {
35        if(u>v) swap(u,v);
36        q.pb((Query){DEL, u, v});
37        int prev = last[ii(u,v)];
38        match[prev] = sz(q)-1;
39        match.pb(prev);
40    }
41    void query() {//podria pasarle un puntero
        donde guardar la respuesta
42        q.pb((Query){QUERY, -1, -1}), match.pb
        (-1);}
43    void process() {
44        forn(i,sz(q)) if (q[i].type == ADD &&
        match[i] == -1) match[i] = sz(q);
45        go(0,sz(q));
46    }
47    void go(int l, int r) {
48        if(l+1==r){
49            if (q[l].type == QUERY)//Aqui
        responder la query usando el dsu!
50            res.pb(dsu.comp);//aqui query=
        cantidad de componentes
        conexas
51            return;
52        }
53        int s=dsu.snap(), m = (l+r) / 2;
54        forr(i,m,r) if(match[i]!=-1 && match[i]<l
        ) dsu.merge(q[i].u, q[i].v);
55        go(l,m);
56        dsu.rollback(s);
57        s = dsu.snap();
58        forr(i,l,m) if(match[i]!=-1 && match[i]>=
        r) dsu.merge(q[i].u, q[i].v);
59        go(m,r);
60        dsu.rollback(s);
61    }
62 }dc;

```

8. Network Flow

8.1. Dinic

```

1
2 const int MAX = 300;
3 // Corte minimo: vertices con dist[v]>=0 (del
        lado de src) VS. dist[v]==-1 (del lado del
        dst)
4 // Para el caso de la red de Bipartite Matching (
        Sean V1 y V2 los conjuntos mas proximos a src
        y dst respectivamente):
5 // Reconstruir matching: para todo v1 en V1 ver
        las aristas a vertices de V2 con it->f>0, es
        arista del Matching
6 // Min Vertex Cover: vertices de V1 con dist[v]
        ==-1 + vertices de V2 con dist[v]>0
7 // Max Independent Set: tomar los vertices NO
        tomados por el Min Vertex Cover
8 // Max Clique: construir la red de G complemento
        (debe ser bipartito!) y encontrar un Max

```



```

// Min Edge Cover: tomar las aristas del matching
+ para todo vertices no cubierto hasta el
momento, tomar cualquier arista de el
int nodes, src, dst;
int dist[MAX], q[MAX], work[MAX];
struct Edge {
    int to, rev;
    ll f, cap;
    Edge(int to, int rev, ll f, ll cap) : to(to),
        rev(rev), f(f), cap(cap) {}
};
vector<Edge> G[MAX];
void addEdge(int s, int t, ll cap){
    G[s].pb(Edge(t, sz(G[t]), 0, cap)), G[t].pb(
        Edge(s, sz(G[s])-1, 0, 0));}
bool dinic_bfs(){
    fill(dist, dist+nodes, -1), dist[src]=0;
    int qt=0; q[qt++]=src;
    for(int qh=0; qh<qt; qh++){
        int u =q[qh];
        forall(e, G[u]){
            int v=e->to;
            if(dist[v]<0 && e->f < e->cap)
                dist[v]=dist[u]+1, q[qt++]=v;
        }
    }
    return dist[dst]>=0;
}
ll dinic_dfs(int u, ll f){
    if(u==dst) return f;
    for(int &i=work[u]; i<sz(G[u]); i++){
        Edge &e = G[u][i];
        if(e.cap<=e.f) continue;
        int v=e.to;
        if(dist[v]==dist[u]+1){
            ll df=dinic_dfs(v, min(f, e.cap-e
                .f));
            if(df>0){
                e.f+=df, G[v][e.rev].f-=
                    df;
                return df; }
        }
    }
    return 0;
}
ll maxFlow(int _src, int _dst){
    src=_src, dst=_dst;
    ll result=0;
    while(dinic_bfs()){
        fill(work, work+nodes, 0);
        while(ll delta=dinic_dfs(src, INF))
            result+=delta;
    }
    // todos los nodos con dist[v]!=-1 vs los que
    // tienen dist[v]==-1 forman el min-cut
    return result; }

```

8.2. Edmonds Karp's

```

1 #define MAX_V 1000
2 #define INF 1e9
3 //special nodes
4 #define SRC 0
5 #define SNK 1
6 map<int, int> G[MAX_V]; //limpiar esto
7 //To add an edge use
8 #define add(a, b, w) G[a][b]=w
9 int f, p[MAX_V];
10 void augment(int v, int minE){
11     if(v==SRC) f=minE;
12     else if(p[v]!=-1){
13         augment(p[v], min(minE, G[p[v]][v]));
14         G[p[v]][v]-=f, G[v][p[v]]+=f;
15     }
16 }
17 ll maxflow(){//O(VE^2)
18     ll Mf=0;
19     do{
20         f=0;
21         char used[MAX_V]; queue<int> q; q.push(SRC);
22         zero(used), memset(p, -1, sizeof(p));
23         while(sz(q)){
24             int u=q.front(); q.pop();
25             if(u==SNK) break;
26             forall(it, G[u])
27                 if(it->snd>0 && !used[it->fst])
28                     used[it->fst]=true, q.push(it->fst), p[
29                         it->fst]=u;
30             augment(SNK, INF);
31             Mf+=f;
32         }while(f);
33         return Mf;
34     }

```

8.3. Max Matching

```

1 int LEFT, r[MAXV]; bool seen[MAXV]; VI AdjList[
    MAXV];
2 bool can_match(int u) {
3     for (auto & v : AdjList[u]) {
4         if (!seen[v]) {
5             seen[v] = true;
6             if (r[v] < 0 || can_match(r[v])) {
7                 r[v] = u; return true;
8             }
9         }
10    } return false;
11 }
12 int max_matching() {
13     memset(r, -1, sizeof r);
14     int ans = 0;
15     for (int u=0 ; u<LEFT ; u++) {
16         memset(seen, 0, sizeof seen);
17         if (can_match(u)) ans++;
18     } return ans;
19 }

```

8.4. Min-cost Max-flow

```

1  const int MAXN=10000;
2  typedef ll tf;
3  typedef ll tc;
4  const tf INFFLUJO = 1e14;
5  const tc INFCOSTO = 1e14;
6  struct edge {
7      int u, v;
8      tf cap, flow;
9      tc cost;
10     tf rem() { return cap - flow; }
11 };
12 int nodes; //numero de nodos
13 vector<int> G[MAXN]; // limpiar!
14 vector<edge> e; // limpiar!
15 void addEdge(int u, int v, tf cap, tc cost) {
16     G[u].pb(sz(e)); e.pb((edge){u,v,cap,0,cost});
17     G[v].pb(sz(e)); e.pb((edge){v,u,0,0,-cost});
18 }
19 tc dist[MAXN], mnCost;
20 int pre[MAXN];
21 tf cap[MAXN], mxFlow;
22 bool in_queue[MAXN];
23 void flow(int s, int t) {
24     zero(in_queue);
25     mxFlow=mnCost=0;
26     while(1){
27         fill(dist, dist+nodes, INFCOSTO); dist[s] =
28             0;
29         memset(pre, -1, sizeof(pre)); pre[s]=0;
30         zero(cap); cap[s] = INFFLUJO;
31         queue<int> q; q.push(s); in_queue[s]=1;
32         while(sz(q)){
33             int u=q.front(); q.pop(); in_queue[u]=0;
34             for(auto it:G[u]) {
35                 edge &E = e[it];
36                 if(E.rem() && dist[E.v] > dist[u] + E.
37                     cost + 1e-9){ // ojo EPS
38                     dist[E.v]=dist[u]+E.cost;
39                     pre[E.v] = it;
40                     cap[E.v] = min(cap[u], E.rem());
41                     if(!in_queue[E.v]) q.push(E.v),
42                         in_queue[E.v]=1;
43                 }
44             }
45             if (pre[t] == -1) break;
46             mxFlow +=cap[t];
47             mnCost +=cap[t]*dist[t];
48             for (int v = t; v != s; v = e[pre[v]].u) {
49                 e[pre[v]].flow += cap[t];
50                 e[pre[v]^1].flow -= cap[t];
51             }
52         }
53     }
54 }

```

9. Template y Otros

Template

```
1 //touch {a..m}.in; tee {a..m}.cpp < template.cpp
```

```

2  #include <bits/stdc++.h>
3  using namespace std;
4  #define forr(i,a,b) for(int i=(a); i<(b); i++)
5  #define forn(i,n) forr(i,0,n)
6  #define sz(c) ((int)c.size())
7  #define zero(v) memset(v, 0, sizeof(v))
8  #define forall(it,v) for(auto it=v.begin();it!=v.
9      end();++it)
10 #define pb push_back
11 #define fst first
12 #define snd second
13 typedef long long ll;
14 typedef pair<int,int> ii;
15 #define dforn(i,n) for(int i=n-1; i>=0; i--)
16 #define dprint(v) cout << #v<" << v << endl //;
17 const int MAXN=100100;
18 int n;
19
20 int main() {
21     freopen("input.in", "r", stdin);
22     ios::sync_with_stdio(0);
23     while(cin >> n){
24
25     }
26     return 0;
27 }

```

Rellenar con espacios(para justificar)

```

1  #include <iomanip>
2  cout << setfill('␣') << setw(3) << 2 << endl;

```

Aleatorios

```

1  #define RAND(a, b) (rand()%(b-a+1)+a)
2  srand(time(NULL));

```

Doubles Comp.

```

1  const double EPS = 1e-9;
2  x == y <=> fabs(x-y) < EPS, x > y <=> x > y +
3  EPS
4  x >= y <=> x > y - EPS

```

Expandir pila

```

1  #include <sys/resource.h>
2  rlimit rl;
3  getrlimit(RLIMIT_STACK, &rl);
4  rl.rlim_cur=1024L*1024L*256L;//256mb
5  setrlimit(RLIMIT_STACK, &rl);

```

Iterar subconjunto

```
1 for(int sbm=bm; sbm; sbm=(sbm-1)&bm)
```

Split

```

1  vector<string> split(string str,string sep){
2      char* cstr=const_cast<char*>(str.c_str());
3      char* current;

```

```
4     vector<string> arr;
5     current=strtok(cstr,sep.c_str());
6     while(current!=NULL){
7         arr.push_back(current);
8         current=strtok(NULL,sep.c_str());
9     }
10    return arr;
11 }
```